Explorable Web Apps to Teach AI to Non-Majors*

Justin Li
Computer Science and Cognitive Science
Occidental College
Los Angeles, CA 90041
justinnhli@oxy.edu

Abstract

We report on the experience of using web apps to teach AI to students with no programming experience. These apps allow students to explore methodological limitations and modeling assumptions, and provide them with algorithmic thinking experience. We conclude with qualitative student feedback and general observations about this approach.

1 Introduction

Computational methods are increasingly important across many disciplines in the physical and social sciences. While many fields simply ask students to visualize the results of machine learning, other fields have a more substantial intersection with CS. Cognitive science students, for example, may be interested in topics such as natural language processing (NLP), human-computer interaction (HCI), and artificial intelligence (AI). Similarly, economics students may look at algorithmic approaches to game theory, and urban planners may want to create agent-based models of sociological phenomenon.

Guest-lecturing to and co-teaching these non-majors present a unique pedagogical challenge. Psychology or economics students may have no experience

in programming, and thus lack the prerequisite knowledge to complete traditional CS assignments such as large coding projects. At the same time, these interdisciplinary collaborations are an opportunity to introduce *computational thinking* [4] to non-majors. This raises the question of what pedagogical approaches could be applied to meet the needs of these non-CS students.

This paper reports on our experience teaching cognitive science students using three custom-built, interactive, "explorable" web apps, including a course catalog prerequisite extractor, an pattern-matching chatbot, and a Bayesian network calculator. We deployed these web apps in co-taught and cross-listed courses between computer science and cognitive science, at both the introductory and advanced levels. The source code for these apps and their associated assignments are available on GitHub[1], with hosted versions on Heroku[2].

These apps present graphical user interface (GUI) that allows the students to engage in algorithmic thinking and problem solving without writing code. These apps not only demonstrate the topic under discussion, but with guided questions, additionally provide three benefits:

1. allow students to discover the limits of a particular approach
2. allow students to examine modeling assumptions
3. provide an algorithmic problem solving experience

Below, we describe how each app operates, a sketch of the associated assignments, and how they provide some of the benefits listed above.

2 Course Catalog Prerequisite Extraction

The prerequisite extraction app was originally created for an upper-level, cross-listed AI course, where only half the students have any CS experience. As part of the module on NLP and information retrieval, students were given a course catalog and asked to extract the prerequisites of each course onto separate lines, as a department code and a course number. Students were allowed to use any programming language of their choice or, for non-technical students, to use the prerequisite extraction app. The app must therefore allow students to manipulate text in a repeatable way without writing code.

To achieve this goal, students use a GUI to specify how the course catalog should be processed (Figure 1). Students can add transformation rules that select and split lines of text, as well as insert, delete, and replace words as necessary. Once the transformations are specified, the app applies them to a

[1] `https://github.com/justinnhli/ccsc-sw-2019-apps`
[2] `https://ccsc-sw-2019-apps.herokuapp.com/`

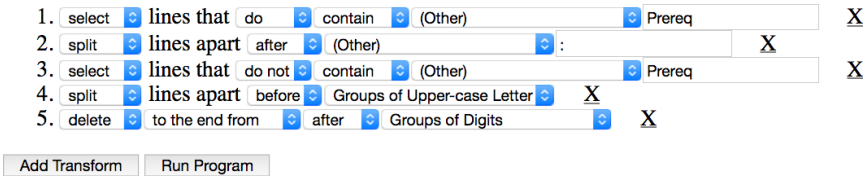

Figure 1: The Course Catalog Prerequisite Extraction app interface.

copy of the course catalog, and shows the original course descriptions and the extracted text side-by-side. This provides immediate feedback for students, who may then modify their transformation rules to fine-tune the extraction.

While students are often quickly able to extract the sentence with prerequisites, the details eventually make perfect extraction difficult. Differences between "CS 101 and 102," "CS 101 and CS 102," and "Computer Science 101, 102" force students to create robust sequences of transformations. Other descriptions such as "any CS 100-level course except CS 130" require additional semantics and data processing to match the required output format. Once students are satisfied with their outputs, they are asked to score their "program" and identify failure cases. The assignment therefore forces students to confront the variability of language, and to experience the challenges both of creating algorithms and of evaluating the performance of an AI.

The interactive app and the questions together demonstrate Benefits 1, 2, and 3. First, despite the seeming generality of the transformations, it is impossible to perfectly extract all prerequisites. Even with a tedious number of transformations to deal with all wording variations, the rules would still fail on negations and other edge cases. The accompanying questions on evaluation further emphasize the limits of such an approach (Benefit 1).

Students are also led to discover misconceptions in their understanding of the domain (Benefit 2). Through this exercise, they realize that course catalogs are not as uniform as they might expect, and even minor variations such as extra commas may cause their programs to fail. For cognitive science students in particular, this may be their first encounter with the difficulty of NLP.

Finally, although students did not write any code, they nonetheless solved a problem with algorithmic/computational thinking (Benefit 3). Much like writing code, students were given the goal of extracting prerequisites, and had to assemble smaller computational building blocks in the appropriate order to achieve the desired result. In essence, the app's GUI provides a domain-specific language for information extraction, without the potential for syntax errors and typos. Through this app, non-CS students can get a sense of the problem solving that programmers engage in.

3 Pattern-Matching Chatbot

The pattern-matching chatbot app was originally created for a guest lecture on the relationship between AI and gender. Mimicking one of the first chatbots, ELIZA, it simply tries to match each received message against a list of patterns, and selects a random response from the first matching pattern. The app was later adapted for a co-taught introductory cognitive science course, as well as advanced courses in both AI and HCI, both of which were cross-listed. Again, the majority of students using the app had no CS experience.

To illustrate the operation of the chatbot, the app provides both a chat window and a textbox where students can edit the patterns and responses. The patterns are indicated by lines that begin with "`@`", while possible responses are indicated by "`%`". The chatbot is also able to match variable text, which are represented by repeated capital letters (e.g. "`XXXXX`") usable in both the patterns and the responses. Whenever the patterns change, they are dynamically loaded into the chatbot, so students have immediate feedback on how the new patterns affect the flow of the conversation.

The accompanying instructions for the app depend on whether the students are absolute beginners or more advanced students. Beginners are simply asked to modify the patterns to remove the mystery of how chatbots might work. For advanced students, this app served as a leaping-off point for deeper discussion of conversational interfaces. By attempting to design a useful chatbot with this limited architecture, students see how semantics, conversational context, and even timing may be necessary for human-level language understanding.

The chatbot app demonstrates Benefits 1 and 3. Although it is obvious that pattern-matching is fragile, students may not be able to articulate the exact scenarios in which the approach breaks down. By placing students in the chat, the app allows students to evaluate the effectiveness of the chatbot (Benefit 1). Advanced students designing chatbots are also forced to consider the effect of patterns being matched in order, and to structure the patterns to provide both specific and generic responses. While the pattern-and-response inputs to the chatbot app are less expressive than that of the prerequisite extraction app, it nonetheless asks students to consider how computational units work together to create the desired output (Benefit 3).

4 Bayesian Network Calculator

The Bayesian network app was created for the cross-listed AI course. Bayesian networks are a graphical model for reasoning about causality and the probability of occurrence of events. In addition to being commonly used in AI and robotics, it has also gained traction as a general modeling tool in the sciences.

This app allows students to visualize a Bayesian network and calculate probabilities. Students enter the causal relationships between events and the conditional probability tables in a textbox, which the app visualizes graphically. Students can then further specify observations of events or to calculate the posterior probability based on those observations.

This app demonstrates Benefits 1 and 2. Since students are asked to create and evaluate a Bayesian model of their choice, they must justify the causal relationships and the conditional probabilities, which often requires data that students do not have (Benefit 1). Students are then prompted for cases where the model gives unintuitive results and to explain the discrepancy — whether the network fails to model the phenomenon due incorrect causality/probabilities, or if it is their intuition that is faulty. These questions and the app together provoke students to critique their models and potentially discover insight about the phenomenon, much as real modelers might (Benefit 2).

5 Qualitative Student Feedback

Student evaluations and feedback over the past three years provide informal evaluation of these interactive apps as pedagogical tools. For example, in the reflection component of the assignment, a student wrote that the prerequisite extraction app provided the *"challenge of figuring out what worked and what didn't work,"* which hints at the algorithmic problem solving required. Another student described how the *"huge amount of variability in how people convey information which makes it difficult to correctly isolate desired information,"* and that as a result, *"coding for information retrieval must be tailored specifically to the corpus."* These comments suggest that the app was effective in provoking examination of the current limits of the NLP. Similarly, a student reflected on their Bayesian network and how they *"didn't think about how those probabilities would interact with each other"*, a failure in their model of the phenomenon that led to discrepancies between the prediction and their intuition. In general, the feedback suggest that these open-ended web apps engaged students in critically assessing the AI techniques. Further study may be able to quantify the effect on student learning compared to other activities.

6 Comments and Conclusion

Interactive pedagogical material is not new. Multiple digital CS textbooks now contain auto-graded coding exercises, which have been shown to improve test scores as compared to static textbooks [1]. Others have applied the lab science paradigm to teach AI, with the goal of increased engagement and providing

experiential learning to students [2].

What differentiates the apps in this paper is the focus on non-CS students. We draw inspiration from "explorable explanations", online documents that combine a narrative with interactive "illustrations" [3]. Such documents emphasize accessibility to a broad audience by preferring graphical interfaces, and often explore a combinatorial space where the user can make self-guided discoveries. Although the web apps presented in this paper are designed with supporting instruction and guided questions for the classroom context, the focus on accessibility is maintained. Explorable explanations have influenced other parts of the design of these web apps. All of the apps have textual inputs (or have inputs directly encoded into the URL), which makes the result easily sharable for collaboration. The open-ended input of text transformation rules and Bayesian network structure aims to make discoveries possible, and it naturally leads to discussions of limitations and modeling assumptions. All three apps are also relatively domain independent, and can by used by non-major students to engage in computational thinking in their domain of expertise.

To the best of our knowledge, neither explorable explanations nor educational web apps have received much discussion in the literature. This paper reported our experience with three different interactive web apps, their design, their use in coursework, and feedback from students hinting at the learning outcomes. In our experience, the presentation of CS content to non-CS students have benefited from use of interactive web apps that allow students to explore CS concepts without writing code. We anticipate that as conventions develop and as supporting frameworks mature, these approaches for non-CS pedagogy will receive more attention and its benefits more formally studied.

References

[1] Alex Daniel Edgcomb *et al.* Student performance improvement using interactive textbooks: A three-university cross-semester analysis. In *2015 ASEE Annual Conference & Exposition*, Seattle, Washington, 2015.

[2] Stephanie Elizabeth August. Enhancing expertise, sociability, and literacy through teaching artificial intelligence as a lab science. In *2012 ASEE Annual Conference & Exposition*, San Antonio, Texas, 2012. ASEE Conferences.

[3] Bret Victor. Explorable explanations. `http://worrydream.com/ExplorableExplanations/`, 2011.

[4] Jeannette M. Wing. Computational thinking. *Communications of the ACM*, 49(1):33–35, 2006.